Redes Multiservicio

Práctica 3: Implementación de un servicio telemático basado en webcam (1.5 ptos)

Objetivos

- *Desarrollar una aplicación en JAVA que transmita una imagen capturada por una webcam desde un PC a otro PC remoto*
- *Manejar los sockets para enviar y recibir información a través de los protocolos TCP y UDP*

Tareas a realizar

La realización de esta práctica consiste en desarrollar una aplicación en JAVA que permita enviar una señal multimedia desde un PC a otro a través de la red. La información multimedia a transmitir consistirá en una imagen capturada por una webcam a través de la aplicación. Dicha aplicación seguirá el modelo cliente-servidor para la comunicación entre los dos PCs.

Las tareas a realizar son las siguientes:

1.- (0.2 ptos.) Instalar el JDK de JAVA, una API de manejo de webcams en JAVA y un IDE para desarrollar la aplicación (ver sección: "A. Instalación del software necesario").

2.- (0.3 ptos.) Desarrollar el código necesario para un cliente y un servidor que permitan el envío de una imagen capturada por webcam mediante el uso de sockets TCP o de tipo *Stream* (ver sección: "B. Servicio telemático mediante sockets TCP").

3.- (0.5 ptos.) Repetir la tarea anterior pero esta vez mediante el uso de sockets UDP o de tipo datagrama (ver sección: "C. Servicio telemático mediante sockets UDP").

4.- (Opcional) Escoger y resolver alguno de los siguientes retos:

- Convertir el servidor de la tarea 2 en un servidor concurrente mediante el uso de hilos (*threads*) de forma que el servidor lance un nuevo hilo de ejecución para gestionar la conexión de cada cliente nuevo que se conecta (ver sección: "D. Ejemplo de servidor concurrente").

- Añadir a la funcionalidad desarrollada en la tarea 3 la inteligencia necesaria para una comunicación fiable: retransmisiones, confirmaciones (ACK), reordenado o espaciado (*pacing*) de paquetes.

Nota: se recomienda realizar las tareas en primer lugar sobre un mismo PC utilizando la dirección "localhost" y posteriormente entre dos PCs diferentes usando la red del laboratorio.

A. Instalación del software necesario

Para la realización de la práctica es necesaria la instalación del JDK de JAVA. Se trata de un software que incluye un conjunto de paquetes básicos para el desarrollo de cualquier aplicación en JAVA. Se recomienda descargar la última versión: https://www.oracle.com/java/index.html.

Igualmente, para facilitar la tarea de programación, se recomienda el uso de un entorno integrado de programación (IDE), como por ejemplo, Netbeans: https://netbeans.org/.

En relación con el manejo de webcams, es necesario descargarse una librería (o API) que facilite las tareas al programador. En particular, este software se puede descargar desde: http://webcam-capture.sarxos.pl/. Posteriormente, deberá descomprimirlo en una carpeta.

Una vez que haya creado un nuevo proyecto en el entorno integrado de programación, deberá agregar los paquetes de la librería de manejo de webcams. Para ello, agregue el archivo de extensión ".jar" ubicado dentro de la carpeta de esta librería y aquellos ubicados dentro de la subcarpeta "libs".

B. Servicio telemático mediante sockets TCP

Para familiarizarse con la API de webcams, en primer lugar, se recomienda probar algunos ejemplos incluidos en la carpeta "examples" de la librería, en particular, "TakePictureExample.java" y "WebcamPanelExample".

En cuanto a la implementación del cliente y el servidor utilizando sockets TCP, se recomienda revisar la teoría que se explicó en el seminario previo. A continuación, se muestra un ejemplo de cliente TCP, el cual se conecta al servidor FTP (puerto 21) y visualiza la primera línea que recibe del servidor:

```
import java.net.*;
import java.io.*;

class ClienteTCP {
      public static void main(String args[]) throws
                                        UnknownHostException, IOException {
            Socket s=new Socket("ugr.es",21);
            BufferedReader in=new BufferedReader(new
                                    InputStreamReader (s.getInputStream()));
            System.out.println(in.readLine());
            s.close();
      }
}
```

El siguiente ejemplo corresponde a un servidor que espera un cliente. Cuando éste se conecta, el servidor le envía un mensaje de bienvenida y acaba:

```
import java.net.*;
import java.io.*;

class ServidorTCP {
      public static void main(String args[]) throws
                                          UnknownHostException,IOException {
            ServerSocket ss=new ServerSocket(7777);
            Socket s=ss.accept(); // espero a que llegue un cliente
            PrintWriter out=new PrintWriter(s.getOutputStream(),true);
            out.println("Bienvenido a la asignatura RMS");
            s.close();
            ss.close();
      }
}
```

Para representar la imagen recibida en una ventana puede hacer uso de la clase “JFrame”.

C. Servicio telemático mediante sockets UDP

A modo de ejemplo, el siguiente código implementa un cliente UDP, el cual envía un datagrama a un servidor e imprime la respuesta:

```
import java.net.*;
import java.io.*;

public class ClienteUDP{
      public static void main(String[] args) throws IOException {
            DatagramSocket s = new DatagramSocket();
            InetAddress dir =
                        InetAddress.getByName("ugr.es");
            String msg = "Hola, soy un alumno\n";
            byte[] buf = new byte[256];
            buf = msg.getBytes();
            DatagramPacket p = new DatagramPacket(buf, buf.length,
                                                dir, 7777);
```

```
            s.send(p);
            s.receive(p); // se bloquea hasta recibir un datagrama
            System.out.write(p.getData());
            s.close();
      }
}
```

El siguiente código es un ejemplo de servidor UDP, el cual envía de vuelta el datagrama recibido, sin modificarlo, a la dirección IP y puerto de origen. Una vez que procesa un cliente, acaba:

```
import java.net.*;
import java.io.*;
public class ServidorUDP{
      public static void main(String[] args) throws IOException {
            DatagramSocket s = new DatagramSocket(7777);
            DatagramPacket p = new DatagramPacket(new byte[256], 256);
            s.receive(p); // se bloquea hasta que recibe un datagrama
            p.setAddress(p.getAddress());
            p.setPort(p.getPort());
            s.send(p);
            s.close();
      }
}
```

Al igual que en los ejemplos, en esta tarea deberá desarrollar una clase para el cliente y otra para el servidor. Se recomienda seguir la siguiente organización en estas clases:

Cliente:

```
public class ClienteUDP {
      public static void main(String []argumentos){
            int puerto=8767;
            String direccionServidor="127.0.0.1";
            // por si queremos especificar la dirección IP y el puerto por
                                                      la línea de comandos
            if(argumentos.length==2){
                  direccionServidor=argumentos[0];
                  puerto=Integer.parseInt(argumentos[1]);
```

```
            }
            // Creamos un cliente
            new ClienteUDP(direccionServidor,puerto);
      }
      // Constructor de esta clase
      private ClienteUDP(String direccionServidor, int puerto) {
      /*    1. Creamos el socket
            2. Enviamos un mensaje al servidor para pedir la imagen
            3. Recibimos el número de datagramas que se va a recibir
            4. Recibimos y almacenamos la información del servidor
            5. Creamos un stream de bytes ByteArrayInputStream para ir
               leyendo la información recibida
            6. Leemos el primer dato que es el número de filas
            7. Leemos el segundo dato que es el número de columnas
            8. Copiamos los pixeles recibidos a un objeto imagen
            9. Representamos la imagen    */
      }
```

Servidor:

```
public class ServidorUDP {
      public static void main(String []argumentos){
            int puerto=8767;
            // por si queremos especificar el puerto por la línea de
                                                                  comandos
            if(argumentos.length>0){
                  puerto=Integer.parseInt(argumentos[0]);
            }
            // Creamos una objeto de esta clase, pasando como argumento el
                                              puerto donde debe escuchar
            new ServidorUDP(puerto);
      }
      // Constructor de esta clase
      private ServidorUDP(int puerto) {
      /*    1. Creamos el socket
            2. Esperamos un mensaje
            3. Almacenamos dir. IP y puerto en el datagrama a enviar
            4. Abrimos la cámara y capturamos una imagen
            5. Copiamos la imagen a un stream (ByteArrayOutputStream)
               Para ello, añadimos primero el número de filas (getHeight),
               luego el número de columnas (getWidth) y por último los
               pixeles RGB (getRGB)
```

```
            6. Almacenamos en un array de bytes la información a
               transmitir
            7. Transmitimos el número de datagramas que se van a enviar
            8. Transmitimos los datagramas que contienen los datos de la
               imagen (se recomienda usar la función sleep entre cada
               envío) */
        }
```

Los siguientes métodos son útiles para la conversión entre tipos de datos:

```
// Funcion que convierte un array de bytes a un entero
public static int byteArrayToInt(byte[] b) {
  if (b.length == 4)
      return b[0] << 24 | (b[1] & 0xff) << 16 | (b[2] & 0xff) << 8 | (b[3] & 0xff);
  else if (b.length == 2)
      return 0x00 << 24 | 0x00 << 16 | (b[0] & 0xff) << 8 | (b[1] & 0xff);
  else if (b.length == 1)
      return b[0];
  else
      return 0;
  }
}


// Funcion que convierte un entero a bytes
public static byte[] toBytes(int i)
  {
  byte[] resultado = new byte[4];

  resultado[0] = (byte) (i >> 24);
  resultado[1] = (byte) (i >> 16);
  resultado[2] = (byte) (i >> 8);
  resultado[3] = (byte) (i /*>> 0*/);

  return resultado;
  }
}
```

D. Ejemplo de servidor concurrente

El siguiente código muestra un ejemplo de servidor concurrente, el cual envía a cada cliente el tiempo actual en milisegundos:

```
import java.net.*;
import java.io.*;
class SCTCP extends Thread {
    Socket id;
    public SCTCP(Socket s) {id=s;}
    public void run() {
           try {
                 PrintWriter out=new
                 PrintWriter(id.getOutputStream(),true);
                 while(true){
                   out.println(System.currentTimeMillis());
                   sleep(100);
                 }
           } catch(Exception e) {}
    }
    public static void main(String args[]) throws IOException{
           ServerSocket ss=new ServerSocket(8888);
           while(true) new SCTCP(ss.accept()).start();
           }
     }
```